GraphEA: A 3D Educational Tool for Genetic Algorithms

Rafael Dinis
Institute Polytechnic of Coimbra
ISEC - Institute of Engineering
Rua Pedro Nunes, 3030-199
Coimbra, Portugal
a21180313@alunos.isec.pt

Anabela Simões
Institute Polytechnic of Coimbra
ISEC - Institute of Engineering
Rua Pedro Nunes, 3030-199
Coimbra, Portugal
abs@isec.pt

Jorge Bernardino
Institute Polytechnic of Coimbra
ISEC - Institute of Engineering
Rua Pedro Nunes, 3030-199
Coimbra, Portugal
jorge@isec.pt

ABSTRACT
During the last decades Genetic Algorithms (GAs) have proved to be a powerful technique for solving difficult problems. Consequently, GA courses are becoming increasingly common in universities. The laboratorial classes of such courses are crucial for students to consolidate and apply the concepts learned in theoretical classes. However, it is required a lot of programming effort and sometimes students tend to have difficulties on this part, either because the number of different GA variants they have to implement or even because the lack of programming skills. To overcome this problem we present a new educational tool for GAs called GraphEA. This tool aims to help students to learn GAs without the need of programming effort, offering novel features like the 3D visualization of the chromosome formation process and the online modification of problem data. In this paper we demonstrate three well-known optimization problems implemented on the tool, namely the Knapsack Problem, the Traveling Salesman Problem, and the Function Optimization Problem.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education – *Computer science education.*

Keywords
Genetic algorithms; educational and visualization tools; evolutionary computation

1. INTRODUCTION
Genetic Algorithms (GAs) are computational models inspired by the Darwinian principles of evolution of the species and the laws of genetics. Therefore, they can be considered a computational approach to an adaptive system. They are probabilistic algorithms that provide a form of parallel search for solutions to a given problem. GAs were introduced by John Holland [16] in the 1960s at the University of Michigan.

At the biological level, every organism has a set of "rules" that describe how it is built since the early development of life. These rules are encoded in *genes*. These genes are grouped in long chains called *chromosomes*, where each gene represents a specific aspect of an organism, such as eye color or hair color. Thus, each of these aspects can have a fairly large set of configurations or combinations. Genes and their possible settings are called *genotype* and the physical organism, that is, its visible characteristics, are called the *phenotype*. When two organisms come together, they share their genes, generating one (or more) child(ren). These children have genes from both parents that prompted it. This process is called *crossover*, but occasionally one (or more) of these genes can be *mutated*. Generally, the mutation does not make changes on the development of a *phenotype*. But occasionally, a change may dictate a completely new way in the evolution of those organisms. Another important biological concept is the selection. The selection ensures that the fittest organisms are more likely to survive and can thus generate a greater number of offspring than less fit organisms. This allows the best characteristics of individuals to propagate through the generations.

Genetic Algorithms are a computational implementation of the ideas forth above. These algorithms apply *changes* on the elements of the *search space* in order to find high quality solutions to a problem. The set of candidate solutions to a problem is called *population* and each element of the population, that is, a possible solution to the problem, is called individual or chromosome. The components of each chromosome are called genes. The position of a gene in a chromosome is called *locus* and the set of all values that a gene can take is called *alleles*. At the beginning of the execution of the algorithm, an *initial population* is randomly generated and the mechanisms of evolution are successively applied to the elements/chromosomes of the population. The applied transformations are the selection of the best elements, to serve as parents, and the application of *genetic operators* to generate new solutions. These genetic operators consist mainly in the implementation of the mechanisms of *crossover* and *mutation*, inspired by the natural evolution. A genetic algorithm is defined by discrete moments of evolution of the chromosomes, called *generations*. Throughout these generations a commitment must be maintained between the quality of these chromosomes and their diversity. A GA is therefore an iterative process that ends when it reaches a certain *stopping criterion*. This criterion can be, for example, the establishment of a maximum number of generations for evolution. In addition to the mechanisms of selection and the genetic operators, we must take into account two crucial aspects that affect the success of a GA. These aspects are the choice of the *representation* for the solutions of the search space, and the definition of an *evaluation function* that calculates the degree of adaptation of the chromosome/solution for a specific problem, giving a certain quality.

Since Holland's first ideas [16], Genetic Algorithms became popular and efficient mechanisms to solve a wide variety of real problems, and are used in several areas, such as Economics,

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.

Finance, Health, Security or Engineering. As a consequence, an increasingly number of university courses has included GA courses in their curricular programs. The practical classes of GA courses are essential to consolidate the concepts learned in theoretical classes and generally involve a lot of programming effort. Instructors generally ask students to implement many different variations of GAs, and some difficulties begin to arise on students. The main difficulty is the lack of time to implement and make experiments on all the different GA variants, but sometimes the lack of programing skills can also bring difficulties and make the students to lose their interest on this field. To overcome this problem and to allow students and teachers to have an interactive and attractive approach to explore different variants of GAs, a new educational tool was developed. This paper describes this tool, which shows in a novel way the operation of the various components of a Genetic Algorithm. The tool can be explored using three different optimization problems: the Knapsack Problem (KSP), the Traveling Salesman Problem (TSP) and the Function Optimization Problem (FOP). The tool has innovative features like the 3D visualization of the chromosome formation process, and an intuitive interface that explains each step of the evolutionary process of a GA. Furthermore, the tool allows the online modification of problem data and the online algorithm parameterization, so the students can make their own experiences and improve their knowledge on the field of GAs. The developed tool is called GraphEA and is freely available online at [9].

The requirements considered behind the development of GraphEA were based on the didactic features that are needed by GA educational tools for a better learning experience and for motivating students. These requirements are the capability of online parameterization of the GA; the capability of online modification of problem data; the level of user-interaction; the fast access to didactic context information; the number of different problems addressed. Most of the existent GA educational tools have some lack of the above-mentioned requirements. Therefore, the GraphEA tool tries to cover this gap, and improve the state-of-the-art on this field.

This paper is organized as follows: Section 2 reviews the related work made on the field of GA educational tools, taking into account the above mentioned requirements. Then, Section 3 describes the GraphEA tool, presenting the main features of the tool. Section 4 explains the different optimization problems implemented on the tool. Section 5 exposes a feature comparison between the GraphEA tool and the similar tools analyzed on Section 2. Finally, Section 6 draws the main conclusions about the work herein presented and sets topics for future work.

2. RELATED WORK

One of the first attempts to promote the learning of GAs can be pointed to the work made by Stasko [27]. Stasko introduced in 1989 a tool called *TANGO* for promoting the research of algorithm animation. This tool implements a permutation based GA for the Traveling Salesman Problem and allows the visualization of both phenotype and genotype data. Specifically, it allows the visualization in a city diagram of the best-so-far solution found by the GA, as well as the content of the chromosomes of the population. This tool also offers a simple visualization of the chromosomes formation process. However, despite this tool offers a great improvement in algorithm representation, it was not directly geared to teach GAs, and users can have difficulties in understand the concepts behind the scenes during the execution of the GA.

A tool called *SPLICER* [28] was developed at NASA to be used for GA research offering a *Graphical User Interface* (GUI) to parameterize and evaluate the results of GAs using independent problem encoding. Its implementation is based on a modular design, which allows its extension for additional GA functionality, namely the addition of representation libraries, fitness modules, and user interface modules. Independently of the representation library or fitness module being used, this tool allows the online parameterization of the GA. Also, similarly to the TANGO, this tool implements a GA for the Traveling Salesman Problem, which allows the visualization in a city diagram of the best-so-far solution, found by the GA. A disadvantage of this tool is that it was not directly planned to promote GAs education and it is assumed that the user already has knowledge on GAs to make experiences and to implement additional functionalities.

The GA visualization tool developed by Collins and Routen [4] started the early research of graphical representations for showing the fitness and constitution of chromosomes. Three types of genotype visualization were developed on this tool, namely the *overlaid chromosome icons*, the *population bar charts*, and the *allele versus locus frequency matrices*. This tool has introduced novel graphical representations for chromosomes, but it does not allow the online parameterization of the GA.

In the same year of 1993, Kapsalis et al. proposed an educational tool named *GAmeter* [17]. This tool implements a binary coded GA and allows the graphical visualization of the fitness and content of the chromosomes. Moreover, it is possible to make the online parameterization of the GA as well as the control of the algorithm execution. A novel feature provided by this tool is the possibility to arbitrarily edit the contents of a chromosome in the population.

Spears [26] introduced a set of five visualization tools for exploring a GA based on binary coding. Two of these tools were developed for the graphical visualization of the fitness of the chromosomes, namely the *2D Fitness Landscape* and the *3D Fitness Landscape*. The three other tools were developed for chromosome visualization, namely the *Pixel Oriented Visualization*, the *2nd Order Schemata*, and the *Parent Color Stripping*. The last one offers an interesting way to visualize the content of a chromosome, by seeing which genes came from which parent. Similarly to the tool developed by Collins and Routen, these tools provided new ways to graphically represent chromosomes, but they do not allow the online parameterization of the GA.

Dabs and Schoof [6] developed in 1995 an educational tool called *Giga* that implements a permutation based on GA for the Traveling Salesman Problem, and offers a set of windows to visualize the content of the chromosomes. One of the windows allows the visualization of the application of the crossover and mutation operators on the chromosomes. More characteristics of this tool are the control of the execution of the GA, and its online parameterization. Like the TANGO tool, *Giga* allows the visualization, in a city diagram, of the best solution find so far by the GA.

Collins [5] presented in 1998 an educational tool called *GONZO* that implements a binary coded GA for the *MaxInt* problem, allowing the graphical visualization of the fitness and content of the chromosomes. Additionally, this tool allows the user to control the algorithm, and to make parameterizations during its execution. The author introduces three different views on this tool: the *Fitness Versus Time Graph* view, the *Search Space*

Visualization view, and the *Fine-grained Chromosome* view. A very interesting feature of *GONZO* is to allow the user to evaluate the solutions found by the GA, becoming him/herself the evaluation function of the algorithm.

Wu et al. [31], in 1996, proposed an *offline* analysis tool called *VIS,* which allows the detailed analysis of a GA's run. This tool offers five different graphical representations for the chromosomes, namely: *Genotype*, *Zebra*, *Neapolitan*, *Four Color*, and *Gene Location* representations. Also, it is offered the possibility to view the parents of a given chromosome, as well as its corresponding children. As the name implies, a drawback of this tool is that it does not allow any configuration of the GA during its execution.

In 2001, Hart and Ross [13] introduced another *offline* analysis tool called *GAVEL*. This tool allows the analysis of a GA's run in a particular way: at the end of the execution it displays the history of the formation of the best chromosome found by the GA. The representation strategy used for this feature is the *ancestry tree*. This tool offers three graphical representations for the chromosomes, namely: *Alleles Values*, *Gene Origins*, and *Operator Origins*. Like the *VIS* tool, this tool has the disadvantage of not allowing any parameterization of the GA during its execution.

Parejo et al. [25] introduced a framework for meta-heuristic optimization called *FOM*, which includes an implementation of the Traveling Salesman Problem and allows the visualization of the fitness of the chromosomes. This framework has the drawback of not allowing the online parameterization of the GA.

Liao and Sun [32] developed an educational tool called *EGALT*, which allows the parameterization of a binary coded GA for function optimization, and the visualization of the respective results. Besides the GA basic configuration, this tool offers two interesting features which are the visualization of the best chromosome found so far by the GA, and the visualization of its optimization speed, measured in genes per second (g/s). A screenshot of this tool is shown in Figure 1 (a).

Andreas Kerren [1] proposed a visualization tool called *EAVis*, based on a binary coded GA for the Knapsack Problem. This tool allows the control of the execution of the GA, its online parameterization, and the graphical visualization for the fitness and content of the chromosomes. A great innovation of *EAVis* is a window called *3D Fitness Evolution* that offers a three-dimensional visualization of the fitness of the chromosomes. However, this three-dimensional view allows only seeing the fitness of the chromosomes, but not their content itself. The main window of this tool is illustrated in Figure 1 (b).

Brownlee [3] proposed a tool named *OAT* for making experiments on optimization algorithms, including GAs. This tool implements GAs for the Traveling Salesman Problem and the Function Optimization Problem, as well as offers a city diagram for displaying the best-so-far solution for the TSP. Despite the intuitive interface of this tool, it has the drawback of not allowing the online parameterization of the GAs.

Ventura et al. [29] developed a Java-based framework for evolutionary computation called *JCLEC* to serve as a basis for future implementations in evolutionary algorithms. This tool offers a GUI called *GenLab*, which allows to control and to parameterize any algorithm, implemented using this framework. Also, this GUI allows observing the fitness of the individuals along the algorithm's execution. The authors also implemented a binary coded based GA on this framework to serve as a case study for their work. A drawback on this framework is that it does not allow the visualization of the content of the chromosomes during the evolutionary process.

A framework for heuristic and evolutionary algorithms named *HeuristicLab* was proposed by Wagner [30], which permits the parameterization of GAs for the Knapsack Problem, the Traveling Salesman Problem and Function Optimization Problem, as well as the visualization of the fitness of the chromosomes. This tool offers a very rich and intuitive interface and also allows the visualization of the best-so-far solution found by the GA for the TSP on a cities diagram. A drawback of this framework is that all the parameterizations must be done offline.

Luke et al. [21] presented a framework for Evolutionary Computation research named *ECJ*. This framework allows a highly flexible configuration of the implemented algorithms, including GAs. However, it does not allow the online parameterization of the GAs, and offers a user interface too much focused on scientific research, which makes it difficult to use by users without GA knowledge.

Lukasiewycz et al. [22] proposed another framework for evolutionary computation named *Opt4J*. This framework implements a GA for the KSP, which allows the basic configuration, and visualization of the fitness of the chromosomes. A drawback of this framework is that it does not allow the online parameterization of the GA.

Kronfeld et al. [19] presented a framework for heuristic optimization named *EvA2, which* implements the Knapsack Problem and the Traveling Salesman Problem, as well as allows the visualization of the fitness of the chromosomes. A city diagram shows the best-so-far solution found by the GA of the

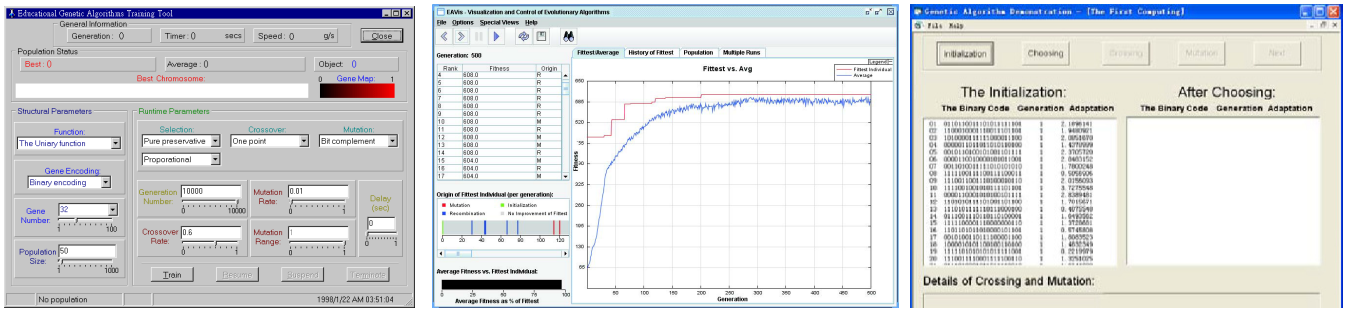

Figure 1. (a) The EGALT tool; (b) The EAVis tool; (c) Educational tool developed by Li and Zhang

TSP. This tool also has the drawback of not allowing the online parameterization of the GA.

More recently, Li and Zhang [15] developed an educational tool to teach GAs, using the Traveling Salesman Problem as optimization problem. The tool gives the student the possibility to configure the GA and to analyze the respective results. Moreover, it allows the step-by-step visualization of the genetic evolution of the individuals. Although this tool only offers the basic configuration for the GA, it allows the user to interact with the evolutionary process, being able to show the genetic content of the individuals after the crossover and mutation operators being applied. A screenshot of this tool is presented in Figure 1 (c).

All the tools herein presented have their strengths and weaknesses. Although many of them implement some of the requirements mentioned in section 1, none of them offers the possibility to visualize in 3D the contents of the chromosomes and their respective formation process.

In contrast, the tool presented in this paper introduces a new concept of 3D visualization of the data processed by GAs and provides a new way for students to learn GAs without using programming.

3. THE GraphEA TOOL

The main goal of the *GraphEA* tool is to graphically show the full power of the genetic algorithms. Features like the parameterization of the genetic algorithm and the 3D visualization of the constitution and formation of chromosomes intend to give the user an enjoyable experience, not only to learn the components of the genetic algorithms, but also to allow a detailed analysis of the effectiveness of the operators and other mechanisms inherent to these algorithms.

Concerning the implemented algorithms, the choice relied on three of the most important optimization problems - the Knapsack Problem, the Traveling Salesman Problem and the Function Optimization Problem. These problems use respectively three different types of representation for the solutions: *binary*, *permutations* and *real-valued*. Using these three types of representations, GraphEA provides the opportunity to explore different aspects related to specific representations, such as different genetic operators and how these operators act on the chromosomes of the algorithm.

Concerning didactic features, the GraphEA tool offers for each implemented GA the following capabilities:

- The continuous monitoring of the quality of the chromosomes: average quality of the generation, best chromosome of the generation, and best chromosome ever;
- The possibility to control the execution of the algorithm, and to parameterize any aspect of the algorithm during execution: selection, elitism, population size, number of generations, crossover operator and respective probability, mutation operator and respective probability, among other functionalities;
- A novel 3D visualization of the population (see Figure 2), allowing the user to navigate through a 3D ambient and to select an arbitrary chromosome and visualize its genetic content, as well as the effects of application of the crossover and mutation operators over the genes (see Figure 3).

The interaction of the user with the Traveling Salesman Problem and the Functions Optimization Problem was enhanced with components that allow to view and to explore the data used by the genetic algorithm. In particular, for the Traveling Salesman Problem, the *Cities Diagram* allows the user to view the layout of the cities of the problem, create, move and delete cities and also see the best route found by the algorithm during its execution. This feature is shown in Figure 4 (a). For the Functions Optimization Problem, the *Graphic Plotter* allows the display of the function that the algorithm is optimizing, and shows the minimum or maximum point found by the algorithm, if the problem is minimization or maximization, respectively. This feature is shown in Figure 4 (b). The Graphic Plotter can only display functions where the number of variables is one or two. Also, for this problem, the function's compiler allows the user to introduce an arbitrary function with one or two real variables. The user can also adjust the domains of these variables, and the Graphic Plotter constantly represents the function.

The GUI of the tool is structured into four main sections that are displayed and numbered in Figure 2. These sections have associated the following functionalities:

1. *Problem Selection Panel* – This panel allows the choice of the problem to display on the interface. The problem selected on this panel determines the content that is presented on the other components of the interface. The parameterization of the selected algorithm can be made using the sub-panels that compose it. The "Know more" link next to each sub-panel gives access to further information about the parameterization that is allowed on that sub-panel;
2. *Chromosome Evaluation Chart* – This chart plots the fitness of the chromosomes of the problem that is currently being viewed. The green line exposes the quality of the fittest chromosome ever found by the algorithm. The red line gives the quality of the best chromosome of the current generation. The blue line represents the average quality of the population of the current generation;
3. *3D Ambient* – This section allows the three-dimensional visualization of the population of the algorithm that is currently selected. Each object on the ambient is a chromosome of the population. For each generation the ambient is refreshed with the child chromosomes belonging to that generation. To view the formation process of a chromosome it is just necessary to click on it, and the *Chromosome Viewer* window (Figure 3) will be exposed with such information;
4. *Algorithm Execution Control Bar* – This control bar gives the full control of the execution of the selected algorithm. Pressing the leftmost button, if the algorithm is running, it stops it immediately and eliminates the population. The center button allows to run, to pause and to resume the execution of the algorithm. The track bar on the right allows the adjustment of the speed of the algorithm. This speed is expressed as a percentage of the real speed that the computer can run the algorithm.

Summarizing, the advantages offered by this tool in comparison to the tools already existent are the following:

- A 3D GUI for each optimization problem, showing the evolutionary process of the population in an interactive way;
- A 2D Cities Diagram for the TSP problem, offering the student the possibility to view the best solution found so far by the

GA, and to interactively configure the disposition of the cities, even during the execution of the GA;

- The 3D visualization of the best solution found so far by the GA of the Function Optimization Problem on a Graphic Plotter;
- The ability to the student create its own optimization function for the Function Optimization Problem (up to 2 variables) and to configure the domain of such variables;
- A great number of parameterization controls for the GA of each optimization problem, including the selection method, the stopping criterion, the population size, the crossover operator and the mutation operator;
- An easy way to access detailed information about the implemented genetic mechanisms; always is available a link on the GUI near each parameterization control to allow the student to access further information about such component of the GA;
- The possibility to export optimization results for future analysis and comparisons.

4. OPTIMIZATION PROBLEMS ADDRESSED BY THE GraphEA TOOL

This section describes the three optimization problems implemented on the proposed educational tool. These problems are respectively the *Knapsack Problem*, the *Traveling Salesman Problem*, and the *Function Optimization Problem*. For each problem it will be presented its definition, the representation adopted for the chromosomes, the evaluation function, and the respective implemented genetic operators.

In GAs, the selection method is independent from the problem but assumes an important role in the algorithm's performance. GraphEA provides two different selection methods that can be chosen for every problem: *Tournament* and *roulette-wheel*. A detailed description of these methods can be found in [9].

4.1 The Knapsack Problem

The *Knapsack Problem* has been studied for more than one hundred years and it is one of the most popular combinatorial optimization problems, being considered an NP-hard problem. A formal definition for this problem can be presented as follows [20]:

> *Given a set I with N objects, each one characterized by a weight W(i) and a value V(i), where i=1,...,N, and a knapsack with capacity C, the objective is to optimize a set S, subset of I, such that the maximum capacity of the knapsack C is not exceeded.*

The representation adopted for this problem is *Binary Representation*. Each position of the binary chromosome represents an item. If that position (or allele) has the value 0, it indicates that the object was not included in the knapsack. If that position has the value 1, it indicates that the object was included. For example, the following chain of genes belonging to a binary chromosome - {0 0 1 1 0 0 1} - represents the inclusion of the 3rd, 4th and 7th objects in the knapsack. Regarding the evaluation function, since the objective of this problem is to maximize the value of the objects contained on the knapsack, the quality of a chromosome is given by the sum of the value of each object that is part of the solution. However, we must take into account that invalid solutions may appear to evaluate. Thus, it is assigned the value 0 to those solutions. In form of mathematical expression, we can outline the evaluation function for this problem on the following way:

$$Fitness(S) = \begin{cases} \sum_{i=1}^{N} S[i] \times V(Obj_i) & if \sum_{i=1}^{N} S[i] \times W(Obj_i) \leq C \\ 0 & otherwise \end{cases}$$

where S[i] corresponds to the i^{th} allele of the chromosome S. Although this evaluation function takes into account invalid solutions, it does not differentiate the potential that an invalid solution might have. Then, one penalization method and two repair methods are implemented in the tool to safeguard these situations. The penalization method is the *Linear Penalty*, and the repair methods are the *Random Repair* and the *Ratio Repair*. These methods are described in detail at [9]. Concerning the genetic operators, three crossover operators and one mutation operator were developed. The most popular and used crossover operators were implemented: the *Crossover with 1 Cutoff Point*, the *Crossover with N (N>1) Cutoff points*, and the *Uniform Crossover*. Regarding the mutation operators, it was implemented the *Binary Mutation*. All these operators are described in detail at [9].

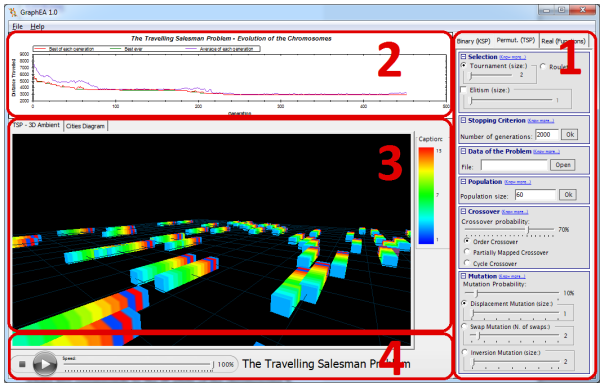

Figure 2. GUI of the GraphEA tool, displaying the 3D ambient for the TSP (Traveling Salesman Problem)

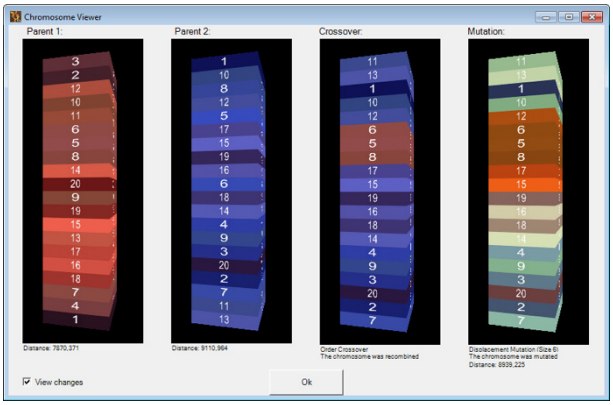

Figure 3. Chromosome Viewer window exposing the formation of a TSP chromosome with 20 genes

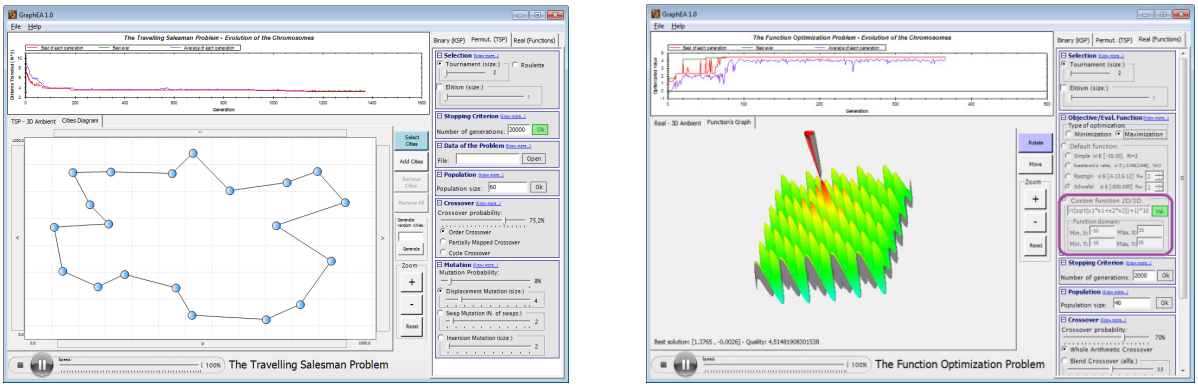

Figure 4. (a) *Cities Diagram* of a TSP with 20 cities; (b) *Graphic Plotter* displaying an arbitrary function entered by the user being maximized by the GA of the *Function Optimization Problem*

4.2 The Traveling Salesman Problem

The *Traveling Salesman Problem* is a combinatorial optimization problem that has been studied since 1930. This problem is also considered an NP-hard problem and can be defined as follows [14]:

> *A traveling salesman has to visit all the cities belonging to its territory and return to its starting point. All visits to the cities are unique. The distance between each pair of cities is known. The objective is to minimize the total distance traveled by the salesman.*

For this problem, the representation used is based on *Permutations*. Each solution, a permutation of N integers, represents the route that the salesman has to travel. The value of N is equal to the number of cities that the salesman has to visit and each city is identified by a different number. The permutation expresses the order of the visits to the cities. An example of a representation of a solution with six cities (identified from 1 to 6) can be: {4,2,1,6,3,5}. Once the adopted representation is quite natural, it can admit invalid solutions: for instance a solution with one repeated city. To avoid the use of penalty or repair algorithms it was decided the development of specific genetic operators to be applied on the chromosomes to preserve the viability of solutions.

The distance traveled in the itinerary that this solution represents measures the quality of a chromosome. Let X be a solution that represents a given permutation of N cities, its quality is given by the following expression:

$$Total\_Distance(X) = Dist\big(X(N), X(1)\big) + \sum_{i=1}^{N-1} Dist\big(X(i), X(i+1)\big)$$

$$Dist(P_1, P_2) = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2}$$

Three crossover operators and three mutation operators were implemented. The implemented crossover operators are order based, this means that no invalid solutions are generated. These crossover methods are the *Order Crossover* proposed by Davis [7], the *Partially Mapped Crossover* proposed by Goldberg and Lingle [8], and the *Cycle Crossover* proposed by Oliver et al. [24]. The implemented mutation operators also assure that the permutations are not disrupted. These mutations operators are the *Displacement Mutation* proposed by Michalewicz [23], the *Swap Mutation* proposed by Banzhaf [2], and the *Inversion Mutation* proposed by Holland [16].

4.3 The Function Optimization Problem

The *Function Optimization Problem* consists on an optimization of a function (minimization or maximization) with continuous variables belonging to a well-defined domain. More formally this problem can be defined as follows [17]:

> *Optimize f(x), Subject to $l \leq x \leq u$, where $x = (x_1, x_2, \ldots, x_i, \ldots, x_n)$ is a vector in R_n, and f(x) is the function to optimize; $l = (l_1, l_2, \ldots, l_i, \ldots, l_n)$ and $u = (u_1, u_2, \ldots, u_i, \ldots, u_n)$ correspond respectively to the domain of the continuous variables, being the domain of x_i defined by $[l_i,u_i]$, and the search space defined by the vectors [l,u].*

The representation adopted for this problem is the *Real-Valued Representation*. A chromosome consists of a vector with floating-point numbers, where each real value of the vector corresponds to a decision variable of the problem. For example, assuming a minimization problem of an arbitrary function with two real variables (x1, x2), both subject to the domain [-1.00, 1.00], a valid solution can be {-0.80, 0.79}. The quality of a chromosome is obtained by assigning the real values of the genes to the variables of the selected objective function. The goal is to obtain the lowest possible value, if we are interested in minimization, or the maximum possible value if the goal is maximization. Since the variables of the function are limited to a predetermined domain, if any of the values of the chromosomes violate that domain, the quality of the chromosome is the worst possible quality. More formally, the quality of a chromosome *X* is given by:

$$Quality(X) = \begin{cases} Objective\_Function(x) & if\ all\ values\ of\ the\ variables\ are\ within\ the\ domain \\ +\infty & otherwise, and\ is\ a\ minimization\ problem \\ -\infty & otherwise, and\ is\ a\ maximization\ problem \end{cases}$$

Regarding the genetic operators for this problem, two crossover operators and two mutation operators were implemented. The implemented crossover operators were the *Whole Arithmetic Crossover* and the *Blend Crossover* and the mutation operators were the *Non-uniform Mutation* and the *Exponential Mutation*. The detailed description of these operators can be found in [9].

The tool allows to choose the function to optimize, using the function compiler, or selecting one of three well-known functions: the *Rosenbrock's Valey Function* [10], the *Rastrigin's Function*, popularized by Mühlenbein [11], and the *Schwefel's Function* [12]. These functions were selected because they have different characteristics (convexity, modality, shape, etc.) allowing the

student to explore the parameterization of the GA and therefore, to gain a better understanding about which aspects of a GA can influence the performance of the function optimization problem.

5. COMPARISION WITH OTHER TOOLS

This section presents a comparison between the features offered by the GraphEA tool and the tools previously described on *Related Work* section. The choice of the characteristics for the comparison was based on two main factors: the capability of parameterization of the tool and its level of user-interaction. This comparison is shown in Table 1. As we can observe, all the features offered by the analyzed tools are also offered by the GraphEA tool with the exception of the feature to modify the value of the genes of a given chromosome. This feature will be implemented on a future version of the tool, where the user will be able to modify a value of an arbitrary gene (for each implemented GA) on the *Chromosome Viewer* window (see Figure 3).

6. CONCLUSIONS AND FUTURE WORK

We have proposed a new tool for learning Genetic Algorithms called *GraphEA* that aims to help students to understand GAs without the need of programming effort. Three well-known optimization problems were implemented in this tool: the Knapsack Problem, the Traveling Salesman Problem, and the Function Optimization Problem. Regarding the main advantages offered by the GraphEA tool in comparison with the already existent tools, we emphasize the following ones:

1. The 3D visualization of the chromosomes constitution as well as their respective history of formation through the application of the crossover and mutation operators;
2. The possibility to modify the problem data during the execution of the GA, namely on the GA for the Traveling Salesman Problem;
3. The easy way to access detailed information about the implemented genetic mechanisms through the context link available on the GUI near each parameterization control.

With these new features we can conclude that the GraphEA tool not only makes an important contribution on the improvement of the state-of-the-art of GA educational tools, but also provides a new enjoyable way for students to learn GAs, and stimulates students to make their own researches on this field. However, some future work needs to be done on the tool, and we can highlight the following points as the next improvements to be implemented:

- Introduction of a world map on the Cities Diagram of the TSP;
- Give the user the possibility to change the value of a gene of an arbitrary chromosome;
- Introduction of a visual component which allows the online visualization and modification of problem data for the KSP;
- Improve the 3D representation of chromosomes, both on the 3D ambient of the population and the detailed visualization of a chromosome.

The GraphEA tool, the user manual and other information are available at [9].

Table 1. A comparison between the features offered by the existent tools and the GraphEA tool

Tool name/author	Online GA parame-terization	Online Statistics	Play/Pause the GA execution	Chromosome visualization	Chromosome history and formation visualization	Gene value modify-cation	Online visualization of problem data	Online problem data modification
GraphEA	Yes	Yes	Yes	Yes (3D)	Yes (3D)	No	Yes	Yes
TANGO	No	No	Yes	Yes	Yes	No	Yes	No
SPLICER	Yes	Yes	Yes	No	No	No	Yes	No
Collins and Routen	No	No	No	Yes	No	No	No	No
GAmeter	Yes	Yes	Yes	Yes	No	Yes	No	No
Giga	Yes	Yes	Yes	Yes	Yes	No	Yes	No
GONZO	Yes	Yes	Yes	Yes	No	No	No	No
VIS	No	No	No	Yes	Yes	No	No	No
GAVEL	No	No	No	Yes	Yes	No	No	No
EGALT	Yes	Yes	Yes	Yes	No	No	No	No
EAVis	Yes	Yes	Yes	Yes	No	No	No	No
FOM	No	Yes	No	No	No	No	No	No
JCLEC	Yes	Yes	Yes	No	No	No	No	No
OAT	No	Yes	No	No	No	No	Yes	No
HeuristicLab	No	Yes	Yes	No	No	No	Yes	No
ECJ	No	Yes	No	No	No	No	No	No
Opt4J	No	Yes	No	No	No	No	No	No
EvA2	No	Yes	No	No	No	No	Yes	No
Li and Zhang	Yes	Yes	Yes	Yes	Yes	No	Yes	No

7. REFERENCES

[1] Andreas Kerren, *Improving Strategy Parameters of Evolutionary Computations with Interactive Coordinated Views*, Proceedings of the 6th IASTED International Conference on Visualization, Imaging, and Image Processing (VIIP '06), ACTA Press, pp. 88-93, 2005.

[2] Banzhaf, W., *The "Molecular" Traveling Salesman*, Biological Cybernetics 64: 7–14, 1990.

[3] Brownlee J., *Oat: the optimization algorithm toolkit*, Tech. rep., Complex Intelligent Systems Laboratory, Swinburne University, 2007

[4] Collins, T. and Routen, T., *Visualization of A.I. techniques*, Proceedings of the International Conference on Computer Graphics and Visualization COMPUGRAPH'93, ACM Press, Portugal, 1993.

[5] Collins, T., *Understanding evolutionary computing: A hands on approach*, in Proceedings of 1998 IEEE International Conference on Evolutionary Computation. Piscataway, NJ: IEEE Press, 1998.

[6] Dabs, T. and Schoof, J., *A graphical user interface for genetic algorithms*, Technical Report 98, Lehrstuhl fur Informatik II, University Wurzburg, DE, 1995.

[7] Davis, L., *Applying Adaptive Algorithms to Epistatic Domains*, Proceedings of the International Joint Conference on Artificial Intelligence, 162–164, 1985.

[8] Goldberg, D. E. and Lingle, R., *Alleles, Loci and the TSP*. In Grefenstette, J. J. (ed.) Proceedings of the First International Conference on Genetic Algorithms and Their Applications, 154–159. Hillsdale, New Jersey: Lawrence Erlbaum, 1985.

[9] *GraphEA Official Website*, [Online], http://www.graphea.pt.vu/, accessed on January 2013.

[10] H. H. Rosenbrock, *An Automatic Method for Finding the Greatest or Least Value of a Function*, The Computer Journal 3: 175–184, 1960.

[11] H. Mühlenbein et al., *The Parallel Genetic Algorithm as Function Optimizer*, Parallel Computing, 17, pp. 619–632, 1991.

[12] H. P. Schwefel, *Numerical Optimization of Computer Models*, John Wiley & Sons, 1981.

[13] Hart, E. and Ross, P., *GAVEL - a New Tool for Genetic Algorithm Visualization*, IEEE Transactions on Evolutionary Computation, Vol. 5(4), pp. 335-348, August 2001.

[14] J. J. Grefenstettte, R. Gopal, B. J. Rosmaita, D. V. Gucht, *Genetic algorithm for the traveling salesman problem*, J. J. Grefenstette, editor, Proceeding of the First International Conference on Genetic Algorithms and Their Applications, pages 160-168, Hillsdale, NJ, July 1985.

[15] Jing Li and Zhaotong Zhang, *A Learning Tool of Genetic Algorithm*, 2010 Second International Workshop on Education Technology and Computer Science, 2010.

[16] John Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press Ann Arbor, 1975.

[17] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. dissertation, University of Michigan,1975.

[18] Kapsalis, A., Rayward-Smith, V., Smith, G., *Fast sequential and parallel implementation of genetic algorithms using the gameter toolkit*, Proceedings of the International Conference on Neural Networks and Genetic Algorithms, Springer-Verlag, Innsbruck, 1993.

[19] Kronfeld, M., Planatscher, H., Zell, A., *The EvA2 optimization framework*, in Blum C, Battiti R (eds) Learning and Intelligent Optimization Conference, Special Session on Software for Optimization (LION-SWOP), Springer, Venice, no. 6073 in Lecture Notes in Computer Science, LNCS, pp 247–250, 2010

[20] L. Cowen, *Lecture 4: The Knapsack Problem*, Comp 260: Advanced Algorithms, Tufts University, Spring 2009.

[21] Luke et al., *Ecj: A java based evolutionary computation research system*, [Online], http://cs.gmu.edu/eclab/projects/ecj/, 2009

[22] Martin Lukasiewycz et al., Opt4j — *The optimization framework for java*, [Online], http://www.opt4j.org, 2009

[23] Michalewicz, Z., *Genetic Algorithms + Data Structures = Evolution Programs*, Berlin Heidelberg: Springer Verlag, 1992.

[24] Oliver, I. M. et al., *A Study of Permutation Crossover Operators on the TSP*, Grefenstette, J. J. (ed.) Genetic Algorithms and Their Applications: Proceedings of the Second International Conference, 224–230. Hillsdale, New Jersey: Lawrence Erlbaum, 1987.

[25] Parejo, J. et al., Fom*: A framework for metaheuristic optimization*, Computational Science ICCS 2003 Lecture Notes in Computer Science 2660:886–895, 2003

[26] Spears, W., *Visualizing genetic algorithms*, Technical Report AIC-94-055, AI Center, Naval Research Laboratory, Washington, D.C., 1994.

[27] Stasko, J., *TANGO: A Framework and System for Algorithm Animation*, PhD thesis, Brown University, Providence, RI, 1989.

[28] Steven E. Bayer and Lui Wang, *Genetic Algorithm Programming Environment: Splicer*, Proceedings of the 1991 IEEE Int. Conf. on Tools for AI, San Jose, California, U.S.A., 1991.

[29] Ventura, S. et al., *JCLEC: a Java framework for evolutionary computation*, Soft Computing - A Fusion of Foundations, Methodologies and Applications - Special issue (pp. 315-357), October 2007.

[30] Wagner, S., *Heuristic Optimization Software Systems - Modeling of Heuristic Optimization Algorithms in the HeuristicLab Software Environment*, PhD Thesis, Institute for Formal Models and Verification, Johannes Kepler University Linz, Austria, 2009.

[31] Wu et al., *Visual analysis of evolutionary algorithms*, Proceedings of the 1999 Conference on Evolutionary Computation (CEC'99), 1999.

[32] Ying-Hong Liao and Chuen-Tsai Sun, *An Educational Genetic Algorithms Learning Tool*, IEEE Transactions on Education, Vol. 44, No. 2, May 2001.